The impact of heterogeneity on master-slave scheduling

Jean-François Pineau, Yves Robert, Frédéric Vivien

The impact of heterogeneity on master-slave scheduling

Jean-François Pineau [a] Yves Robert [a] Frédéric Vivien [a]

[a] *Laboratoire LIP, CNRS-ENS Lyon-INRIA-UCBL, École Normale Supérieure de Lyon, France*

Abstract

In this paper, we assess the impact of heterogeneity on scheduling independent tasks on master-slave platforms. We assume a realistic one-port model where the master can communicate with a single slave at any time. We target both on-line and off-line scheduling problems, and we focus on simpler instances where all tasks have the same size. While such on-line problems can be solved in polynomial time on homogeneous platforms, we show that there does not exist any optimal deterministic algorithm for heterogeneous platforms. Whether the source of heterogeneity comes from computation speeds, or from communication bandwidths, or from both, we establish lower bounds on the competitive ratio of any deterministic algorithm. We provide such bounds for the most important objective functions: the minimization of the makespan (or total execution time), the minimization of the maximum response time (difference between completion time and release time), and the minimization of the sum of all response times. Altogether, we obtain nine theorems which nicely assess the impact of heterogeneity on on-line scheduling. For off-line scheduling, we prove several result for problems with release dates, either optimality or NP-hardness.

These theoretical contributions are complemented on the practical side by the implementation of several heuristics on a small but fully heterogeneous MPI platform. Our results show the superiority of those heuristics which fully take into account the relative capacity of the communication links.

Key words: Master-Slave, Scheduling, On-line algorithm, Off-line algorithm

Email addresses: Jean-Francois.Pineau@ens-lyon.fr (Jean-François Pineau), Yves.Robert@ens-lyon.fr (Yves Robert), Frederic.Vivien@ens-lyon.fr (Frédéric Vivien).

1 Introduction

The main objective of this paper is to assess the impact of heterogeneity on scheduling independent tasks on master-slave platforms. We make some important assumptions that render our following approach very significant in practice. First we assume that the target platform is operated under the one-port model, where the master can communicate with at most one single slave at any time. This model is much more realistic than the standard model from the literature, where the number of simultaneous messages involving a processor is not bounded. Second we mainly consider on-line scheduling problems, i.e., problems where release times and sizes of incoming tasks are not known in advance. Such dynamic scheduling problems are more difficult to deal with than their static counterparts, the off-line problems (for which all task characteristics are available before the execution begins) but they encompass a broader spectrum of applications. But we also study some off-line scheduling problems to assess their difficulty and compare it with their dynamic counterparts.

We endorse the somewhat restrictive hypothesis that all tasks are identical, i.e., that all tasks are equally demanding in terms of communications (volume of data sent by the master to the slave which the task is assigned to) and of computations (number of flops required for the execution). We point out that many important scheduling problems involve large collections, or *bags*, of identical tasks [9, 1]. Without the hypothesis of having same-size tasks, the impact of heterogeneity cannot be studied. Indeed, scheduling different-size tasks on a homogeneous platform reduced to a master and two identical slaves, without paying any cost for the communications from the master, and without any release time, is already an NP-hard problem [13]. In other words, the simplest (off-line) version is NP-hard on the simplest (two-slave) platform.

On the contrary, scheduling same-size tasks is easy on fully homogeneous platforms. Consider a master-slave platform with m slaves, all with the same communication and computation capacity; consider a set of identical tasks, whose release times are not known in advance. We demonstrate in [25, 24] that an optimal approach to solve this on-line scheduling problem is simply to schedule the tasks in the order of their release times, as soon as possible, and to send them to processors in a round-robin fashion. This simple strategy is optimal for many classical objective functions, including the minimization of the makespan (or total execution time), the minimization of the maximum response time (difference between completion time and release time), and the minimization of the sum of the response times.

Off-line and on-line scheduling of same-size tasks on heterogeneous platforms is much more difficult. In this paper, we study the impact of heterogeneity

from two sources. First we consider a communication-homogeneous platform, i.e., where communication links are identical: the heterogeneity comes solely from the computations (we assume that the slaves have different speeds). On such platforms, we show on one hand that there does not exist any optimal deterministic algorithm for on-line scheduling. This holds true for the previous three objective functions (makespan, maximum response time, sum of response times). We even establish lower bounds on the competitive ratio of any deterministic algorithm. For example, we prove that there exist problem instances where the makespan of any deterministic algorithm is at least 1.25 times larger than that of the optimal schedule. This means that the competitive ratio of any deterministic algorithm is at least 1.25. We prove similar results for the other objective functions. On the other hand, we design an optimal makespan minimization algorithm for the off-line problem with release dates. This algorithm generalizes, and provides a new proof of, a result of Simons [31]. The second source of heterogeneity is studied with computation-homogeneous platforms, i.e., where computation speeds are identical: the heterogeneity comes solely from the different-speed communication links. In this context, we prove similar results for on-line scheduling, but with different competitive ratios. But for the off-line problem with release dates, we failed to derive an optimal makespan minimization algorithm, but we provide an efficient heuristic. Not surprisingly, when both sources of heterogeneity add up, the complexity goes beyond the worst scenario with a single source. In other words, for on-line scheduling on fully heterogeneous platforms, we derive competitive ratios that are higher than the maximum of the ratios with a single source of heterogeneity. And we prove that the off-line problem is NP-hard in the strong sense.

The main contributions of this paper are mostly theoretical. However, on the practical side, we have implemented several heuristics on a small but fully heterogeneous MPI platform. Our results show the superiority of those heuristics which fully take into account the relative capacity of the communication links.

The rest of the paper is organized as follows. Section 2 is devoted to an overview of related work. In Section 3, we state some notations and the scheduling problems under consideration. Section 4 is devoted to the theoretical results for on-line problems. We start with a global overview of the approach and a summary of all results. As three platform types and three objective functions lead to nine theorems, we state all our lower bounds, but we only develop the proof of just one of the theorems. The interested reader will find all the remaining proofs in our research report [23]. Section 5 is devoted to off-line problems and heuristics built to solve those problems. We provide an experimental comparison of several scheduling heuristics in Section 6. Finally, we state some concluding remarks in Section 7.

2 Related work

We classify several related papers along the following three main lines:

Models for heterogeneous platforms– In the literature, one-port models come in two variants. In the unidirectional variant, a processor cannot be involved in more than one communication at a given time-step, either a send or a receive. In the bidirectional model, a processor can send and receive in parallel, but at most from a given neighbor in each direction. In both variants, if P_u sends a message to P_v, both P_u and P_v are blocked throughout the communication.

The bidirectional one-port model is used by Bhat et al [8] for fixed-size messages. They advocate its use because "current hardware and software do not easily enable multiple messages to be transmitted simultaneously". Even if non-blocking multi-threaded communication libraries allow for initiating multiple send and receive operations, they claim that all these operations "are eventually serialized by the single hardware port to the network". Experimental evidence of this fact has recently been reported by Saif and Parashar [28], who report that asynchronous MPI sends get serialized as soon as message sizes exceed a few megabytes. Their results hold for two popular MPI implementations, MPICH on Linux clusters and IBM MPI on the SP2.

The one-port model fully accounts for the heterogeneity of the platform, as each link has a different bandwidth. It generalizes a simpler model studied by Banikazemi et al. [2], Liu [19], and Khuller and Kim [16]. In this simpler model, the communication time only depends on the sender, not on the receiver: in other words, the communication speed from a processor to all its neighbors is the same.

Finally, we note that some papers [3, 4] depart from the one-port model as they allow a sending processor to initiate another communication while a previous one is still on-going on the network. However, such models insist that there is an overhead time to pay before being engaged in another operation, so there are not allowing for fully simultaneous communications.

Task graph scheduling– Task graph scheduling is usually studied using the so-called *macro-dataflow* model [21, 30, 10, 12], whose major flaw is that communication resources are not limited. In this model, a processor can send (or receive) any number of messages in parallel, hence an unlimited number of communication ports is assumed (this explains the name *macro-dataflow* for the model). Also, the number of messages that can simultaneously circulate between processors is not bounded, hence an unlimited number of communications can simultaneously occur on a given link. In other words, the communication network is assumed to be contention-free, which of course is not realistic as soon as the processor number exceeds a few units. More recent papers [33, 22, 27, 5, 6, 32] take communication

resources into account.

Hollermann et al. [14] and Hsu et al. [15] introduce the following model for task graph scheduling: each processor can either send or receive a message at a given time-step (bidirectional communication is not possible); also, there is a fixed latency between the initiation of the communication by the sender and the beginning of the reception by the receiver. Still, the model is rather close to the one-port model discussed in this paper.

On-line scheduling– A good survey of on-line scheduling can be found in [29, 26]. Two papers focus on the problem of on-line scheduling for master-slaves platforms. In [**?**], Leung and Zhao proposed several competitive algorithms minimizing the total completion time on a master-slave platform, with or without pre- and post-processing. In [18], the same authors studied the complexity of minimizing the makespan or the total response time, and proposed some heuristics. However, none of these works take into consideration communication costs. To the best of our knowledge, the only previously known results for on-line problems with communication costs are those reported in our former work [25]; in the current paper, we have dramatically improved several of the competitive ratios given in [25] and we have added new ones.

3 Notations and problem formulation

Assume that the platform is composed of a master and m slaves $P_1, P_2, \ldots, P_m$. Let c_j be the time needed by the master to send a task to P_j, and let w_j be the time needed by P_j to execute a task. As for the tasks, we simply number them $1, 2, \ldots, i, \ldots n$. We let r_i be the release time of task i, i.e., the time at which task i becomes available on the master. In on-line scheduling problems, n and the r_i's are not known in advance. Finally, we let C_i denote the end of the execution of task i under the target schedule.

To be consistent with the literature [17], we describe the scheduling problems using the notation $\alpha \mid \beta \mid \gamma$, where each notation represents:

α: the machine environment– As in the standard, we use P for platforms with identical processors, and Q for platforms with different-speed processors. We add MS to this field to indicate that we work with master-slave platforms.

β: the job characteristics– We write r_i when jobs have nonzero release dates, and we add *on-line* for on-line problems, i.e., when the release dates are not known in advance. We write $w_j = w$ for computation-homogeneous platforms (a job running time is independent on the machine executing it), and $c_j = c$ for communication-homogeneous platforms.

γ: the optimality criterion– We deal with three optimality criteria:

- the makespan (total execution time) $C_{max} = \max C_i$;

- the maximum response time (or *max-flow*) $\max F_i = \max(C_i - r_i)$: indeed, $C_i - r_i$ is the time spent by task i in the system;
- the sum of response times $\sum F_i = \sum(C_i - r_i)$ or *sum-flow*, which is equivalent, for complexity considerations, to the sum of completion times $\sum C_i$.

In this paper, we will consider the on-line and off-line versions of problem $Q, MS \mid r_i, w_j, c_j \mid \gamma$, where γ will be either $C_{\max}$, $\max F_i$, or $\sum F_i$.

4 Theoretical results

To assess the performance of an on-line algorithm, one often considers the worst-case relative error between the quality of the computed solution for an instance and the quality of the corresponding optimal solution. An upper bound for the worst-case relative error is called a competitive ratio. Formally, let $f(\mathcal{A}, \mathcal{I})$ denote the objective value, according to the studied optimality criterion, of the schedule produced by algorithm $\mathcal{A}$ on instance $\mathcal{I}$. Then, algorithm $\mathcal{A}$ is ρ-competitive if $f(\mathcal{A}, \mathcal{I}) \leq \rho f(\mathrm{OPT}, \mathcal{I})$ for any instance $\mathcal{I}$, where OPT denote the optimal off-line scheduling algorithm for our objective function.

To prove, for a given objective function, a lower bound on the competitive ratio of any on-line algorithm, we assume a scheduling algorithm and we run it against a scenario elaborated by an adversary. The adversary analyzes the decisions taken by the algorithm, and reacts against them. In the end, we compute the relative performance ratio. The minimum of the relative performance ratios over all execution scenarios is the desired bound on the competitive ratio of the algorithm.

We first prove the bound for the competitive ratio of any algorithm when minimizing makespan on fully heterogeneous platforms.

Theorem 1 *For the problem $Q, MS \mid$ on-line, r_i, w_j, $c_j \mid C_{\max}$, on at least three slaves, there is no scheduling algorithm whose competitive ratio ρ is strictly lower than $\frac{1+\sqrt{3}}{2}$.*

PROOF. Assume that there exists a deterministic on-line algorithm $\mathcal{A}$ whose competitive ratio is $\rho = \frac{1+\sqrt{3}}{2} - \epsilon$, with $\epsilon > 0$. We will build a platform and an adversary to derive a contradiction. The platform is made of three processors P_1, P_2, and P_3 such that $w_1 = \epsilon$, $w_2 = w_3 = 1 + \sqrt{3}$, $c_1 = 1 + \sqrt{3}$ and $c_2 = c_3 = 1$.

Initially, the adversary sends a single task i at time 0. $\mathcal{A}$ executes the task i, either on P_1 with a makespan at least equal to $c_1 + w_1 = 1 + \sqrt{3} + \epsilon$, or on

P_2 or P_3, with a makespan at least equal to $c_2 + w_2 = c_3 + w_3 = 2 + \sqrt{3}$. At time-step 1, we check whether $\mathcal{A}$ made a decision concerning the scheduling of i, and which one:

(1) If $\mathcal{A}$ scheduled the task i on P_2 or P_3, the adversary does not send any other task. The best possible makespan is then $c_2+w_2 = c_3+w_3 = 2+\sqrt{3}$. The optimal scheduling being of makespan $c_1 + w_1 = 1+\sqrt{3}+\epsilon$, we have a competitive ratio of:

$$\rho \geq \frac{2+\sqrt{3}}{1+\sqrt{3}+\epsilon} > \frac{1+\sqrt{3}}{2} - \epsilon,$$

because $\epsilon > 0$ by assumption. This contradicts the hypothesis on ρ. Thus the algorithm $\mathcal{A}$ cannot schedule task i on P_2 or P_3.

(2) If $\mathcal{A}$ did not yet begun to send the task i, the adversary does not send any other task. The best makespan that can be achieved is then equal to $1 + c_1 + w_1 = 2 + \sqrt{3} + \epsilon$, which is even worse than the previous case. Consequently, algorithm $\mathcal{A}$ does not have any other choice than to schedule task i on P_1.

Then, at time-step $\tau = 1$, the adversary sends two tasks, j and k. We consider all the scheduling possibilities:

- j and k are scheduled on P_1. Then, the best achievable makespan is:

  $$\max\{c_1 + 3w_1, \max\{c_1, \tau\} + c_1 + w_1 + \max\{c_1, w_1\}\} = 3(1+\sqrt{3}) + \epsilon,$$

  as we can assume that $\epsilon < \frac{1+\sqrt{3}}{2}$. In other words, the makespan is equal to either:
  - the time needed to send the first task to P_1 plus the execution time of the three tasks;
  - the earliest date at which the second task can be sent, plus the time needed to send the last two tasks and execute them; in turn, this time may be equal to twice the communication time plus once the execution time or, contrary, once the communication time plus twice the execution time.

  (Note that all other expressions in this proof are derived along the same line of reasoning.)
- The first of the two jobs, j and k, to be scheduled is scheduled on P_2 (or P_3) and the other one on P_1. Then, the best achievable makespan is:

  $$\max\{c_1+w_1, \max\{c_1,\tau\}+c_2+w_2, \max\{c_1+2w_1, \max\{c_1,\tau\}+c_2+c_1+w_1\}\}$$

  $$= \max\{1+\sqrt{3}+\epsilon, 3+2\sqrt{3}, \max\{1+\sqrt{3}+2\epsilon, 3+2\sqrt{3}+\epsilon\} = 3+2\sqrt{3}+\epsilon.$$
- The first of the two jobs j and k to be scheduled is scheduled on P_1 and the other one on P_2 (or P_3). Then, the best achievable makespan is:

  $$\max\{c_1+w_1, \max\{\max\{c_1,\tau\}+c_1+w_1, c_1+2w_1\}, \max\{c_1,\tau\}+c_1+c_2+w_2\}$$

$$= \max\{1+\sqrt{3}+\epsilon, \max\{2+2\sqrt{3}+\epsilon, 1+\sqrt{3}+2\epsilon\}, 4+3\sqrt{3}\} = 4+3\sqrt{3}.$$

- One of the jobs j and k is scheduled on P_2 and the other one on P_3. Then, the best achievable makespan is:

$$\max\{c_1+w_1, \max\{c_1,\tau\}+c_2+w_2, \max\{c_1,\tau\}+c_2+c_3+w_3\}$$

$$= \max\{1+\sqrt{3}+\epsilon, 3+2\sqrt{3}, 4+2\sqrt{3}\} = 4+2\sqrt{3}.$$

- The case where j and k are both executed on P_2, or both on P_3, leads to an even worse makespan than the previous case. Therefore, we do not need to study it.

Therefore, the best achievable makespan for $\mathcal{A}$ is: $3+2\sqrt{3}+\epsilon$ (as $\epsilon < 1$). However, we could have scheduled i on P_2, j on P_3, and then k on P_1, thus achieving a makespan of:

$$\max\{c_2+w_2, \max\{c_2,\tau\}+c_3+w_3, \max\{c_2,\tau\}+c_3+c_1+w_1\}$$

$$= \max\{2+\sqrt{3}, \max\{1,1\}+2+\sqrt{3}, \max\{1,1\}+2+\sqrt{3}+\epsilon,\} = 3+\sqrt{3}+\epsilon.$$

Hence, we have a competitive ratio $\rho \geq \frac{3+2\sqrt{3}+\epsilon}{3+\sqrt{3}+\epsilon} > \frac{1+\sqrt{3}}{2} - \epsilon$, which contradicts the hypothesis on ρ. □

Because we have three platform types (communication-homogeneous, computation-homogeneous, fully heterogeneous) and three objective functions (makespan, max-flow, sum-flow), we have nine bounds to establish. Table 1 summarizes the results, and shows the influence of the platform type on the difficulty of the problem. As expected, mixing both sources of heterogeneity (i.e., having both heterogeneous computations and communications) renders the problem the most difficult.

Platform type	Objective function		
	Makespan	Max-flow	Sum-flow
Communication homogeneous	$\frac{5}{4} = 1.250$	$\frac{5-\sqrt{7}}{2} \approx 1.177$	$\frac{2+4\sqrt{2}}{7} \approx 1.093$
Computation homogeneous	$\frac{6}{5} = 1.200$	$\frac{5}{4} = 1.250$	$\frac{23}{22} \approx 1.045$
Heterogeneous	$\frac{1+\sqrt{3}}{2} \approx 1.366$	$\sqrt{2} \approx 1.414$	$\frac{\sqrt{13}-1}{2} \approx 1.302$

Table 1
Lower bounds on the competitive ratio of on-line algorithms, depending on the platform type and on the objective function.

All the results presented in Table 1 are obtained using the same techniques. It would thus be meaningless to detail each of the nine proofs, which the interested reader can find in details in our research report [23]. Below, we just

list the characteristics of the platforms and jobs used to prove all the bounds of Table 1. (As in the proof of Theorem 1, for a lower bound ρ on a competitive ratio, we assume that there exists an algorithm whose competitive ration is $\rho - \epsilon$ for a given positive value of ϵ).

Communication homogeneous platforms

Makespan : The platform consists of two processors of computation times $p_1 = 3$ and $p_2 = 7$, and of communication time $c = 1$. The job arrival times are: 0, 1, and 2.

Sum-flow : The platform consists of two processors of computation times $p_1 = 2$ and $p_2 = 4\sqrt{2} - 2$, and of communication time $c = 1$. The job arrival times are: 0, 1, and 2.

Max-flow : The platform consists of two processors of computation times $p_1 = \frac{2+\sqrt{7}}{3}$ and $p_2 = \frac{1+2\sqrt{7}}{3}$, and of communication time $c = 1$; job arrival times: 0 and $\frac{4-\sqrt{7}}{3}$.

Computation homogeneous platforms

Makespan : The platform consists of two processors of computation time $p = \max\{5, \frac{12}{25\epsilon}\}$, and communication times $c_1 = 1$ and $c_2 = \frac{p}{2}$. The job arrival times are: 0, and three times $\frac{p}{2}$.

Sum-flow : The platform consists of two processors of computation time $p = 3$, and communication times $c_1 = 1$ and $c_2 = 2$. The job arrival times are: 0, and three times 2.

Max-flow : The platform consists of two processors of computation time $p = 2 - \epsilon$, and communication times $c_1 = \epsilon$ and $c_2 = 1$. The job arrival times are: 0, and three times $1 - \epsilon$.

Fully heterogeneous platforms

Makespan : The platform consists of three processors of computation times $p_1 = \epsilon$ and $p_2 = p_3 = 1 + \sqrt{3}$, and of communication times $c_1 = 1 + \sqrt{3}$ and $c_2 = c_3 = 1$. The job arrival times are: 0, and two times 1.

Sum-flow : The platform consists of three processors of computation times $p_1 = \epsilon$, and $p_2 = p_3 = \tau + c_1 - 1$, and of communication times c_1 (any positive value) and $c_2 = c_3 = 1$, where $\tau = \frac{\sqrt{52c_1^2+12c_1+1}-(6c_1+1)}{4}$. The job arrival times are: 0, and two times τ.

Max-flow : The platform consists of three processors of computation times $p_1 = \epsilon$, and $p_2 = p_3 = 3+2\sqrt{2}$, and of communication times $c_1 = 2(1+\sqrt{2})$ and $c_2 = c_3 = 1$. The job arrival times are: 0, and two times 2.

5 Off-line algorithms

Now that we have established lower bounds for all the on-line problems considered, we will study their off-line counterparts, that is the situations where

we know beforehand all the problem characteristics such as the release dates. This study enables us to compare the difficulty of the problems.

5.1 Communication homogeneous platform

In this section, all communications are homogeneous, i.e. $c_j = c$, but processors have different speeds. We order the processors so that P_1 is the fastest processor (w_1 is the smallest computing time w_i), while P_m is the slowest processor.

We aim at designing an optimal algorithm for the off-line version of the problem, with release dates. Our objective is to minimize the *makespan*, i.e. $\max\ C_i$. Intuitively, to minimize the completion date of the last task, it is necessary to allocate this task on the fastest processor (which will finish it the most rapidly). However, the other tasks should also be assigned so that this fastest processor will be available as soon as possible for the last task. We define the greedy algorithm *SLJF (Scheduling Last Job First)* as follows:

Initialization: Take the last task which arrives in the system and allocate it to the fastest processor (Figure 1(a)).

Scheduling backwards: Among the not-yet-allocated tasks, select the one which arrived latest in the system. Assign it, without taking its arrival date into account, to the processor which will begin its execution at the latest, but without exceeding the completion date of the previously scheduled tasks (Figure 1(b)).

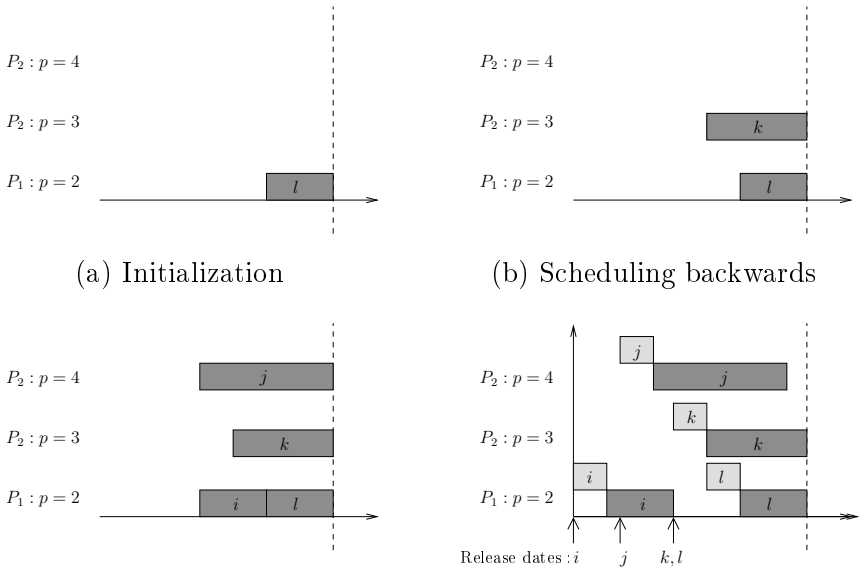
Memorization: Once all tasks are allocated, record the assignment of the tasks to the processors (Figure 1(c)).

Assignment: The master sends the tasks as soon as possible according to their arrival dates, and to the processors which they have been assigned to in the previous step (Figure 1(d)).

Theorem 2 SLJF *is an optimal algorithm for the makespan minimization on communication-homogeneous master-slave platforms, that is for the problem* $Q, MS \mid r_j, w_j, c_j = c \mid C_{\max}$.

PROOF. The first three phases of the SLJF algorithm are independent of the release dates, and only depend on the number of tasks which will arrive in the system. The proof proceeds in three steps. First we study the problem without communication costs, nor release dates. Next, we take release dates into account. Finally, we extend the result to the case with communications. The second step is the most difficult.

For the first step, we have to minimize the makespan when scheduling identical



(a) Initialization (b) Scheduling backwards

(c) Memorization (d) Assignment

Figure 1. Different steps of the SLJF algorithm, with four tasks i, j, k, and l.

tasks on heterogeneous processors and without release dates. Without communication costs, this is a well-known load balancing problem, which can be solved by a greedy algorithm [7]. The "scheduling backwards" phase of *SLJF* follows this greedy algorithm and thus solves this load balancing problem optimally.

Next we add the constraints of release dates. We denote by M the makespan achieved. Let P_j be a processor on which the last processed task is completed at the makespan. Let i be the last task executed on P_j whose processing started at its release date (in other words, we have $C_i = r_i + w_j$). Such a task exists as the processing of the first task executed on P_j began at its release date, the scheduling, in the assignment phase, being done under the as-soon-as-possible policy. By definition of task i, processor P_j is never idle between the release date r_i and the makespan M, and is thus used $M - r_i$ time units during that interval. Our original problem is more constrained than the problem of scheduling $n - i + 1$ tasks whose release dates are all equal to r_i. ($n - i + 1$ corresponds to task i plus the tasks released after it.) Therefore our original problem has an optimal makespan greater than or equal to the makespan of this later and simpler problem. The simpler problem is the well-known load-balancing problem we were referring above, and we have seen that the first step of *SLJF* solves it optimally. Furthermore, *SLJF* solves it in an incremental way : the optimal solution for $k + 1$ tasks is built by optimally adding a task to the optimal solution for k tasks. Therefore, *SLJF* optimally load-balanced the last $n - i + 1$ tasks in the first step. As in this step it uses processor P_j during $M - r_i$ time units, the simpler problem of scheduling $n - i + 1$ tasks whose release dates are all equal to r_i has an optimal makespan of M. From

what precedes, our original problem has thus an optimal makespan greater than or equal to M, which is exactly the makespan achieved by *SLJF*, hence its optimality.

Taking communications into account is now easy. Under the one-port model, with a uniform communication time for all tasks and processors, the optimal policy of the master consists in sending the tasks as soon as possible, using a *First Come First Serve* policy. Now, we consider the date at which a task is available on a slave as its *release date* for our previous problem with release dates and without communications. As a task cannot arrives sooner on a slave than this available date, and as our policy is optimal with release dates, *SLJF* is optimal for *makespan* minimization with release dates and homogeneous communications. □

Remark 3 *It should be stressed that, by posing $c = 0$, our approach allows to provide a new proof for a result of Barbara Simons [31].*

Remark 4 *As it only needs to know in advance the total number of tasks, but not their release dates,* SLJF *is also optimal to minimize the* makespan *for on-line problems where the total number of tasks is known beforehand.*

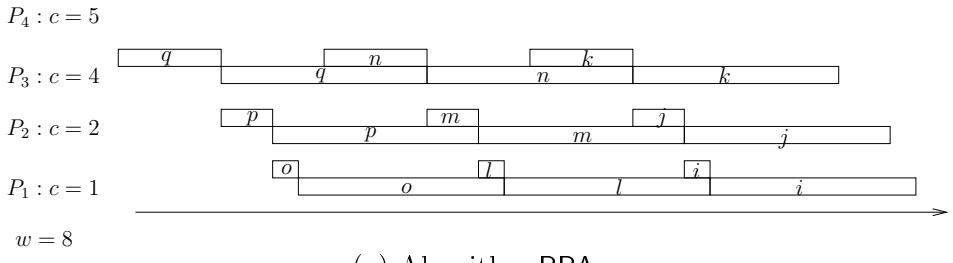
5.2 Computation homogeneous platform

In this section, all the processors are homogeneous, i.e. $w_j = w$, but processor links have different capacities. We order the processors so that P_1 is the fastest communicating processor (c_1 is the smallest of the communication times).

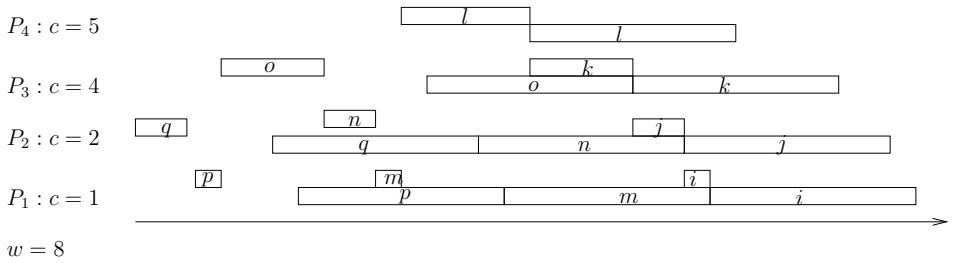
In the easy case where the communication links are fast enough in comparison with the computation time, i.e., $\sum_{i=1}^{m} c_i \leq w$, we can easily prove that the scheduling policy *Round-Robin* which sends by non-increasing communication time and sends the last task on the fastest communication link, is optimal for makespan minimization with release dates. This proof is the same as the one of *SLJF*.

In the general case, as we assume a one-port model, not all slaves will be enrolled in the computation. Intuitively, the idea is to use the fastest m' links, where m' is computed so that the time w to execute a task lies between the time necessary to send a task on each of the fastest $m' - 1$ links and the time necessary to send a task on each of the fastest m' links. Formally,

$$\sum_{i=1}^{m'-1} c_i < w \leq \sum_{i=1}^{m'} c_i$$



(a) Algorithm RRA



(b) Algorithm RRB

Figure 2. Algorithms RRA and RRB with 9 tasks.

With only m' links selected in the platform, we aimed at deriving an algorithm similar to *Round-Robin*. But we did not succeed in proving the optimality of our approach. Hence the algorithm below should rather be seen as a heuristic.

The difficulty lies in deciding when to use the m'-th processor. In addition to be the one having the slowest communication link, its use can cause a moment of inactivity on another processor, if $\sum_{i=1}^{m'-1} c_i + c_{m'} > w$. Our greedy algorithm will simply compare the performance of two strategies, the one sending tasks only on the $m'-1$ first processors, and the other one using the m'-th processor "at the best possible moment".

Let RRA be a *Round-Robin* algorithm, sending the tasks to the $m'-1$ fastest processors, starting with the fastest processor, and scheduling the tasks in the reverse order of release times, from the last one to the first one. In other words, RRA sends the last task to the fastest processor, the previous task to the second fastest processor, and so on. Let RRB be the algorithm sending the last task to processor $P_{m'}$, then following the RRA policy. We see that RRA seeks to continuously use the processors, even though the communication links may sometimes be idle and processor $P_{m'}$ is always idle. On the other hand, RRB tries to reduce the idle time of the communication links by using one more processor at the beginning.

The intuition underlying our algorithm is simple. We know that if we only had the $m'-1$ fastest processors, then RRA would be optimal to minimize the makespan. However, the time necessary for sending a task on each of the $m'-1$ fastest processors is lower than w. This means that sending the tasks takes less time than their execution. This advance, which accumulates over tasks, can become sufficiently large to allow the sending of a task on another m'-th

processor, for "free", i.e., without delaying the treatment of the following tasks on the other processors. Our algorithm can be decomposed into two steps:

- determine i and k, such that $k(m'-1)+i \leq n$, i being the maximum number of tasks that can be sent on m' whose communications can be overlapped by k tasks being sent on every $m'-1$ first processors according to a RRA policy,
- determine the best policy among RRA and RRB to schedule the remaining $n-(k*(m'-1)+i)$ tasks.

We call the resulting greedy heuristic *SLJFWC* for *(Scheduling the Last Job First With Communication)*. Algorithm 1 presents its the pseudo-code version. This heuristic has complexity of $O(n+m\log m)$.

Algorithm 1 Heuristic SLJFWC for problem $Q, MS \mid r_i,\, w_j = w,\, c_j \mid C_{\max}$

$k \leftarrow 0$; $i \leftarrow 0$;
while $k(m'-1)+i \leq n$ and $k\left(\sum_{i=1}^{m'-1} c_i\right) + ic_{m'} + w \leq M$ **do**
  $i \leftarrow i+1$
  $k \leftarrow \left\lceil \frac{ic_{m'}}{w-\sum_{i=1}^{m'-1} c_i} \right\rceil$
end while
compare RRA and RRB with $(n-(k(m'-1)+i))$ tasks
send according to the best scheduling the first $(n-(k(m'-1)+i))$ tasks
send the following $((k(m'-1)+i)-(im'))$ tasks to the $m'-1$ first processors from the slowest to the fastest.
send the last im' tasks to the m' processors from the slowest to the fastest.

5.3 Fully heterogeneous platform

On a fully heterogeneous platform, the problem becomes more difficult as expected. Whereas the problem of minimizing the makespan without release dates can be solved in polynomial time (Section 5.3.1), the general problem with release dates can be proved to be NP-hard (Section 5.3.2).

5.3.1 Without release dates

The problem without release dates can be solved in two ways. The first solution is due to Beaumont, Legrand, and Robert [6] who present an algorithm which, given a platform and n tasks, checks in polynomial time whether one can process the tasks on the platform in a given time T. Using this algorithm, one can find the minimum makespan by performing a binary search on T. As this binary search is over rational values, one could fear that it does not always terminate. One can however show that not only does such a binary search

always terminate, but that it completes in a number of steps polynomial in the size of our problem (the proof is identical to the one used for Theorem 6). This approach leads to a polynomial time algorithm but uses as a building block a complicated algorithm of complexity $O(n^2m^2)$.

To obtain a solution of lower complexity, we propose to reduce our problem to a problem with deadlines. Given a makespan M, we will still check whether there exists a schedule which completes all the work in time. Then, using this as a basic block, we will find the optimal makespan with a binary search. We will use MOORE'S ALGORITHM [20] whose aim is to minimize the number of tardy tasks, i.e., which are not completed by their deadlines. This algorithm gives a solution to the $1||\sum U_j$ problem where the maximum number of tasks, among n candidate tasks, has to be processed in time on a single machine. Each task k, $1 \leq k \leq n$, has a processing time w_k and a deadline d_k before which it has to be processed.

Moore's algorithm —Algorithm 2—, is a classical algorithm in scheduling literature, and works as follows. All tasks are ordered in non-decreasing order of their deadlines. Tasks are added to the solution one by one in this order as long as their deadlines are satisfied. If a task k is out of time, the task j in the actual solution with the largest processing time w_j is deleted from the solution. Moore's algorithm runs in $O(n \log n)$.

Algorithm 2 Moore's algorithm

Input: a set of jobs with their deadlines and sizes: $\{(d_i, w_i)\}_{1 \leq i \leq n}$.
Order the jobs by non-decreasing deadlines: $d_1 \leq d_2 \leq \cdots \leq d_n$
$\sigma \leftarrow \emptyset$; $t \leftarrow 0$
for $i := 1$ to n **do**
  $\sigma \leftarrow \sigma \cup \{i\}$
  $t \leftarrow t + w_i$
  if $t > d_i$ **then**
    Find job j in σ with largest w_j value
    $\sigma \leftarrow \sigma \backslash \{j\}$
    $t \leftarrow t - w_j$
  end if
end for

In order to use Moore's algorithm, which is supposed to schedule tasks with deadlines on one machine, we will need to create tasks with deadlines. Recall we assume a one-port model, and that we only need to schedule the communications, i.e., the use of the communication link by the master (on each worker the assigned tasks are scheduled as soon as possible). Therefore, the communication link corresponds to the machine in Moore's algorithm. For the targeted makespan M, the deadlines and the computation times of the tasks will be defined as follows. We compute for each slave of our platform the deadline of each of the tasks it can receive: d_j^i denotes the deadline of the i-th last task

for the slave P_j. Beginning at the makespan M, one computes when the last task has to arrive on the slave so that it can still be processed in time. The latest moment at which a task can arrive so that it can still be computed on slave P_j is $M - w_j$. Then $d_j^1 = M - w_j$. The latest moment at which the task before the last one can arrive so that it can still be computed in time on slave P_j is $M - 2 \times w_j$. Then $d_j^2 = M - 2 \times w_j$, and so on. We denote by l_j the maximum number of tasks that slave P_j can receive: $l_j = \lfloor \frac{M}{w_j} \rfloor$. See Figure 3 for an example. We denote by l the total number of tasks the slaves could receive: $l = \sum_{i=1}^{m} l_j$. Obviously l is an upper bound and can only be achieved if there are no communication contention, which is unlikely to happen. In other words, with these deadlines we have defined a set of l *virtual* tasks. We are going to effectively schedule as many of these virtual tasks as possible. If the number we succeed to schedule is greater than or equal to the number of tasks we actually have to schedule, n, we will have found a schedule whose makespan is no greater than M. Otherwise we will have failed.

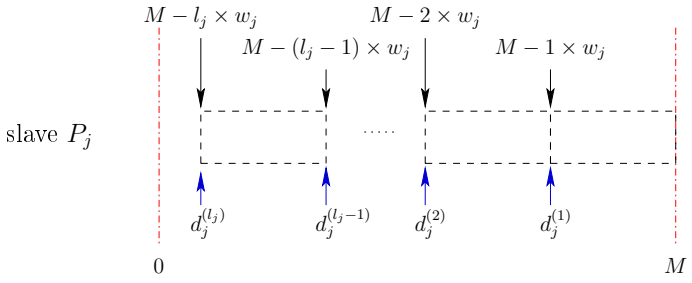


Figure 3. Computation of the deadlines d_j^k for worker P_j.

In our model, the processing time of a task with a deadline d_j^k is the communication time c_j of P_j. Then the master has to decide which virtual tasks have to be sent on which slaves and in which order. To solve this problem we use Moore's algorithm. Starting at time $t = 0$, the master can start scheduling the tasks on the communication link. For this purpose the deadlines d_j^k are ordered by non-decreasing values. In the same manner than in Moore's algorithm, an optimal schedule σ is computed by adding one by one the tasks to the schedule: if we consider the deadline d_j^k, we add a task to processor P_j. So if a deadline is not met, the largest communication is suppressed from σ and we continue. The adapted Moore's algorithm is described by Algorithm 3.

One only needs to add a binary search to finalize our algorithm, which is described by Algorithm 4. We bound the search of the minimal makespan with two values. As at least one machine will have to compute no less than $\frac{n}{m}$ tasks, the lower bound is the minimum time needed by one machine to compute $\frac{n}{m}$ tasks sequentially. Depending on the communication to computation ratio, this time equals to either the time needed to make one communication with the slave and $\frac{n}{m}$ computations, or the time needed to make $\frac{n}{m}$ communications

Algorithm 3 Adapted Moore's algorithm

```
input: {(d_j, c_{i_j})}_j
Order the jobs by non-decreasing deadlines: (d_1, c_{i_1}), ..., (d_l, c_{i_l})
σ ← ∅; t ← 0
for j := 1 to l do
    σ ← σ ∪ {j}
    t ← t + c_{i_j}
    if t > d_j then
        Find job k in σ with largest c_{i_k} value
        σ ← σ\{k}
        t ← t − c_{i_k}
    end if
end for
return σ
```

and one computation. We will call this value f_{min}, and we have:

$$f_{min} = \min_j \left\{ \frac{n}{m} \max\{c_j, w_j\} + \min\{c_j, w_j\} \right\}.$$

The upper bound, f_{max}, is the minimum time needed by one machine to compute all the tasks. We have:

$$f_{max} = \min_j \{n \max\{c_j, w_j\} + \min\{c_j, w_j\}\}.$$

The output of Algorithm 4 is a set σ of couples of deadlines and communication times. From such a set one straightforwardly defines a schedule: the virtual tasks described by σ are sent by the master as soon as possible and by non decreasing deadlines, a virtual task (d_j, c_{i_j}) being sent to the processor P_{i_j}. This schedule has a makespan no greater than M by construction of the deadlines and by the optimality of Moore's algorithm.

Theorem 5 *Algorithm 4 builds an optimal schedule σ for the scheduling problem $Q, MS \mid w_j,\, c_j \mid C_{\max}$.*

PROOF. Algorithm 4 is nothing but a binary search other the makespan M, the core of the algorithm being a call to Algorithm 3 (Adapted Moore's Algorithm) on a set of virtual tasks. Therefore, to prove the optimality of Algorithm 4 we only have to show that Algorithm 3, when called on a virtual set of tasks built for the objective makespan M, outputs a set σ containing at least n tasks if, and only if, $M \geq \mathcal{M}$, where $\mathcal{M}$ is the optimal makespan.

Let us take any value $M \geq \mathcal{M}$. From what precedes, if Algorithm 4 returns a set σ containing at least n elements, there exists a schedule whose makespan is less than, or equal to, M. Conversely, by definition of the optimal makespan

Algorithm 4 Algorithm for problem $Q, MS \mid w_j,\, c_j \mid C_{\max}$

input: $w_i = \frac{\alpha_i}{\beta_i}, \alpha_i, \beta_i \in \mathbb{N} \times \mathbb{N}^*$, $c_i = \frac{\gamma_i}{\delta_i}, \gamma_i, \delta_i \in \mathbb{N} \times \mathbb{N}^*$
$\lambda \leftarrow \mathrm{lcm}_{1 \le i \le m}\{\beta_i, \delta_i\}$
$precision \leftarrow \frac{1}{\lambda}$
$lo \leftarrow \min_j\{\frac{n}{m} \max\{c_j, w_j\} + \min\{c_j, w_j\}\}$
$hi \leftarrow \min_j\{n \max\{c_j, w_j\} + \min\{c_j, w_j\}\}$
$M \leftarrow hi$
repeat
  for $i := 1$ to m **do**
    $l_i \leftarrow \lfloor \frac{M}{w_i} \rfloor$
    for $k := 1$ to $\min\{l_i, n\}$ **do**
      $S \leftarrow S \cup \{(M - k \times w_i, c_i)\}$
    end for
  end for
  if $\sum l_i < n$ **then**
    /* M is too small */
    $lo \leftarrow M$
  else
    $\sigma \leftarrow$ Adapted Moore's algorithm (S)
    if $|\sigma| < n$ **then**
      /* M is too small */
      $lo \leftarrow M$
    else
      /* M is maybe too big */
      $hi \leftarrow M$
      $\sigma \leftarrow \sigma_{\mathrm{opt}}$
    end if
  end if
  $gap \leftarrow |lo - hi|$
  $M \leftarrow (lo + hi)/2$
until $gap < precision$
return σ_{opt}

$\mathcal{M}$, there exists a schedule σ^* of n tasks with a makespan less than, or equal to, M. We will prove that in this case Algorithm 3 returns a set σ containing at least n elements.

Let N_j denote the number of tasks received by P_j under σ^*. So we have $n = \sum_j N_j$. Let us denote by D the set of virtual tasks computed by our algorithm for the scheduling problem for makespan M:

$$D = \bigcup_i \bigcup_{1 \le j \le l_i = \lfloor \frac{M}{w_i} \rfloor} \{(M - j \times w_i, c_i)\}.$$

We also define the set of virtual tasks corresponding to the N_j latest deadlines

in D for each slave P_j:

$$D^* = \bigcup_i \bigcup_{1 \leq j \leq N_i} \{(M - j \times w_i, c_i)\}.$$

Obviously $D^* \subseteq D$. The set of tasks in σ^* is exactly a set of tasks that respects the deadlines in D^*. The application of Moore's algorithm on the same problem returns a maximal solution. With $D^* \subseteq D$, we already know that there exists a solution with $n = |D^*|$ scheduled tasks. So Moore's algorithm will return a solution with $|\sigma| \geq |D^*| \geq n$ tasks (as there may be more than n possible deadlines in D), as we wanted. $\square$

Theorem 6 *Scheduling problem $Q, MS \mid w_j,\ c_j \mid C_{\max}$ is solvable in polynomial time by Algorithm 4.*

PROOF. Thanks to Theorem 5, we only have to show that Algorithm 4 runs in polynomial time.

We perform a binary search for a solution in the interval $[f_{min}, f_{max}]$. As we are in heterogeneous computation conditions, we have heterogeneous w_i-values: for each $i \in [1; m]$, $w_i \in \mathbb{Q}$. The communications are also heterogeneous, so we have $c_i \in \mathbb{Q}$ for each $i \in [1; m]$. For each $i \in [1; m]$, let the representation of the values be of the following form:

$$w_i = \frac{\alpha_i}{\beta_i}, \alpha_i, \beta_i \in \mathbb{N} \times \mathbb{N}^*, \quad \text{and} \quad c_i = \frac{\gamma_i}{\delta_i}, \gamma_i, \delta_i \in \mathbb{N} \times \mathbb{N}^*,$$

where α_i and β_i are relatively prime, and also γ_i and δ_i are relatively prime.

Let λ be the least common multiple of the denominators β_i and δ_i,i.e., $\lambda = \text{lcm}_{1 \leq i \leq m}\{\beta_i, \delta_i\}$. As a consequence, for any i in $[1..m]$, $\lambda \times w_i \in \mathbb{N}$ and $\lambda \times c_i \in \mathbb{N}$. Now we have to choose the precision which allows us to stop our binary search. For this, we take a look at the possible finish times of the workers: all of them are linear combinations of the different c_i and w_i-values. Some optimal algorithms may have some idle times, but without any lost of generality, we only look at the algorithms which send tasks and compute them as soon as possible. So if we multiply all values with λ we get integers for all values and the smallest gap between two finish times is at least 1. So the precision p, i.e., the minimal gap between two feasible finish times, is $p = \frac{1}{\lambda}$.

The maximal number of different values M we have to try can be computed as follows: we examine our algorithm in the interval $[f_{min}, f_{max}]$. The possible values have an increment of $\frac{1}{\lambda}$. So there are

$$\left(\frac{nm - n}{m} \times \min_j\{\max\{c_j, w_j\}\}\right) \times \lambda$$

possible values for M. Hence, the total number of steps of the binary search is

$$O\left(\left(\frac{nm-n}{m}\min_j\{\max\{c_j,w_j\}\}\right)\times\lambda\right).$$

Now we have to prove that this is polynomial in the size of our problem input.

Our platform parameters c_i and w_i are given under the form $w_i=\frac{\alpha_i}{\beta_i}$ and $c_i=\frac{\gamma_i}{\delta_i}$. So it takes $\log(\alpha_i)+\log(\beta_i)$ to store a w_i and $\log(\gamma_i)+\log(\delta_i)$ to store a c_i. So our entry E can be bounded as follows:

$$E\geq\sum_i\log(\alpha_i)+\sum_i\log(\beta_i)+\sum_i\log(\gamma_i)+\sum_i\log(\delta_i).$$

We can do the following estimation:

$$E\geq\sum_i\log(\beta_i)+\sum_i\log(\delta_i)=\log\left(\prod_i\beta_i\times\prod_i\delta_i\right)\geq\log(\lambda).$$

So we already know that our complexity is bounded by

$$O\left(|E|+\log\left(\frac{nm-n}{m}\min_j\{\max\{c_j,w_j\}\}\right)\right).$$

We can simplify this expression:

$$O\left(|E|+\log\left(n\min_j\{\max\{c_j,w_j\}\}-\frac{n}{m}\min_j\{\max\{c_j,w_j\}\}\right)\right)$$
$$\leq O\left(|E|+\log\left(n\min_j\{\max\{c_j,w_j\}\}\right)\right).$$

It remains to upper-bound:

$$\log(n\min_j\{\max\{c_j,w_j\}\})=\log(n)+\log(\min_j\{\max\{c_j,w_j\}\}).$$

n is a part of the input and hence its size can be upper-bounded by the size of the input E. In the same manner we can upper-bound $\log(\min_j\{\max\{c_j,w_j\}\})$ by $\min_j\{\max\{\log(\alpha_j)+\log(\beta_j),\log(\gamma_j)+\log(\delta_j)\}\}\leq E$.

Assembling all these upper-bounds, we get

$$O\left(|E|+\log\left(n\min_j\{\max\{c_j,w_j\}\}\right)\right)\leq O(3|E|)$$

and hence our proposed algorithm needs $O(|E|)$ steps to perform the binary search. Furthermore, the set S contains at most mn elements and the complexity of each call to the Adapted Moore's algorithm is thus $O(mn\log(mn))$, which is polynomial in the size of our problem input [1]. The total complexity finally is $O(|E|\times nm\log nm)$ which is polynomial in the input size. □

[1] We make the usual assumption that our scheduling problem takes the tasks 1,

We can see here a new line between NP-hard problems and easier problems, because the scheduling problem becomes NP-hard when the platform is a heterogeneous tree instead of a star [11].

5.3.2 With release dates

We will prove that the problem $Q, MS \mid r_i \ ; \ w_j \ ; \ c_j \mid C_{max}$ is NP-hard in the strong sense. For that purpose, we will study the following decision problem:

Definition 7 ***MS-hetero:*** *Given a fully heterogeneous master-slave platform, n tasks with release dates, and a deadline D, is it possible to schedule those tasks onto this platform such are all tasks are completed before the deadline D ?*

Those two problems are equivalent. If *MS-hetero* can be solved in polynomial time, then our problem could also be solved in polynomial time using a binary search on D on the interval $[\min_j\{\frac{n}{m} \times \max\{c_j, w_j\} + \min\{c_j, w_j\}\}, r_n + \min_j\{n \times \max\{c_j, w_j\} + \min\{c_j, w_j\}\}]$. Reciprocally, if we can solve our problem and find the minimum makespan D_{opt}, then for all $D \geq D_{opt}$, *MS-hetero* has a schedule, elsewhere it does not.

We practice here a reduction from the work of Dutot [11].

Definition 8 (MS-Dutot) *We suppose a one-port model. Let $T = (V, E)$ be a tree. Let P_{-1} in V be a special vertex called "Master node". For each other vertex, let w_i be the computation cost. For all edges e_i in E, let c_i be the communication cost. Finally let n be a number of tasks and D be a deadline.*

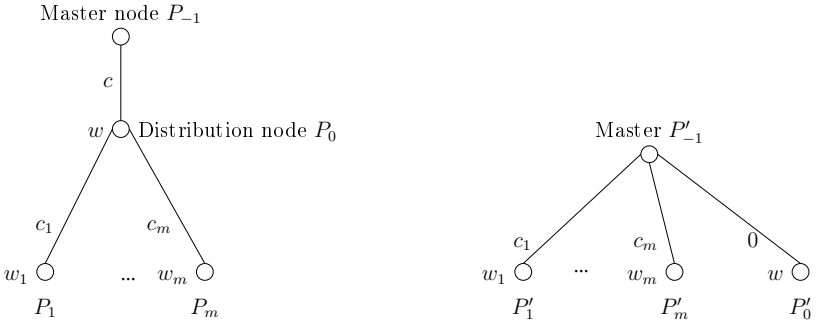
The decision problem is: "Is it possible to schedule the n tasks before the deadline D"?

Theorem 9 ([11]) MS-Dutot *is NP-complete in the strong-sense.*

Theorem 10 MS-hetero *is NP-complete in the strong-sense.*

PROOF. First, *MS-hetero* is in NP. If we have a schedule for n tasks and a given master-slave platform, we can check in polynomial time that this schedule respects the deadline D.

2, ..., n as input, and not merely the number of identical tasks. This is a natural assumption which guarantees that basic certificates, such as sets of task starting times, are of size polynomial in the size of the input, and thus that scheduling problems are in NP.



Platform T used in *MS-Dutot*. Platform T' used in *MS-hetero*.

Figure 4. The platforms used in the reduction of *MS-Dutot* to *MS-hetero*.

Let S be an instance of *MS-Dutot*. S is made of a tree T and of n tasks $J_1, ..., J_n$. We suppose that T is a two-level tree as illustrated on Figure 5.3.2, with one master node P_{-1}, one distribution node P_0, and m slaves, P_1, ..., P_m. P_{-1} is only connected to P_0, and P_0 is connected to all other vertices. (As the platform topology is a tree these are the only edges it contains.) It takes a time w for P_0 to compute a task, and w_i for processor P_i, $1 \leq i \leq m$. It takes a time c for a task to be sent other the edge (P_{-1}, P_0) and, for $i \in [1, m]$, it takes a time c_i on the edge (P_0, P_i).

We can assume without any lost of generality that any instance S has this shape as Dutot exactly used this kind of tree to build his proof of NP-completeness.

From instance S of *MS-Dutot*, we build an instance S' of *MS-Hetero*, composed of a tree T' and n tasks, $J'_1, ..., J'_n$, with their respective release dates, $r'_1, ..., r'_n$.

T' is a fork tree, composed of one master P'_{-1} and $m+1$ leaves $P'_0, P'_1, .., P'_m$. The communication cost of the edge (P'_{-1}, P'_0) is 0 and, for each $i \in [1, m]$, the cost of the edge (P'_{-1}, P'_i) is c_i. The computation cost of P'_0 is w and, for each $i \in [1, m]$, the computation cost of P'_i is w_i. Finally, the release dates of the tasks are uniformly separated: $\forall\ 1 \leq i \leq n, r_i = ic$. This reduction is obviously of polynomial in the size of the original instance.

We now prove the equivalence between problems S and S':

- We first suppose that problem *MS-hetero* on S' has a solution, i.e., that there exists a schedule σ' which execute the n tasks on the $1+m$ processors, according to their release dates, and in an overall time less than or equal to D. From σ', we create a new schedule σ for S. Under σ,the master node P_{-1} sends all the tasks as soon as possible, the i-th task J_i being sent during the time interval $[(i-1)c, ic[$. Therefore, under σ a task J_i arrives on P_0 exactly at the release date of its corresponding task J'_i.

  For any $i \in [1, n]$, J_i is executed on P_j under σ if and only if J'_i is executed on P'_j under σ'. Furthermore, J_i is sent to P_j under σ at the date J'_i is sent

to P'_j under σ', and the two corresponding tasks are executed at the same time. In particular, if J'_i was executed on P'_0, P_0 executes itself J_i without forwarding it to one of its slaves.

As σ' successfully schedules the n tasks with release dates J'_1, ..., J'_n on T' before the deadline D, σ schedules the n J_1, ..., J_n tasks on T before D.

- We now suppose that instance S of *MS-Dutot* has a solution, i.e., that there exists a schedule σ which executes the n tasks J_1, ..., J_n on T before the deadline D. As the time needed by the master P_{-1} to send tasks to the distribution node P_0 is c, task i will arrive on the distribution node at a time t_i greater than, or equal to, r_i.

  Then, symmetrically to what we have done in the previous case, we define σ' from σ as follows: for any $i \in [1, n]$, J'_i is executed on P'_j under σ' if and only if J_i is executed on P_j under σ, J_i is sent to P_j under σ at the date J'_i is sent to P'_j under σ', and the two corresponding tasks are executed at the same time.

  We have noticed that, for any $i \in [1, n]$, $t_i \geq r_i$, therefore schedule σ' respects the release dates. As it has the same makespan than σ, we can conclude.

In conclusion, *MS-hetero* is NP-complete in the strong sense. □

6 MPI experiments

To complement the previous theoretical results, we looked at some efficient on-line algorithms, and we compared them experimentally on different kind of platforms. In particular, we include in the comparison our last two new heuristics, which were specifically designed to work well on communication-homogeneous and on computation-homogeneous platforms respectively.

6.1 Algorithms

We describe here the different algorithms used in the practical tests:

(1) ***SRPT*:** *Shortest Remaining Processing Time* is a well known algorithm on a platform where preemption is allowed, or with task of different size. But in our case, with identical tasks and no preemption, its behavior is the following: it sends a task to the fastest free slave; if no slave is currently free, it waits for the first slave to finish its task, and then sends it a new one.

(2) ***LS*:** *List Scheduling* can be viewed as the static version of *SRPT*. It uses its knowledge of the system and sends a task as soon as possible to the

slave that would finish it first, according to the current load estimation (the number of tasks already waiting for execution on the slave).

(3) ***RR*** **:** *Round Robin* is the simplest algorithm. It simply sends a task to each slave one by one, according to a prescribed ordering. This ordering first chooses the slave with the smallest $w_i + c_i$, then the slave with the second smallest value, etc.

(4) ***RRC*** has the same behavior than *RR*, but uses a different ordering: it sends the tasks starting from the slave with the smallest c_i up to the slave with the largest one.

(5) ***RRP*** has the same behavior than *RR*, but uses yet another ordering: it sends the tasks starting from the slave with the smallest w_i up to the slave with the largest one.

(6) ***SLJF*** is described in a previous section. It is optimal for makespan minimization on communication-homogeneous platform as soon as it knows the total number of tasks.

(7) ***SLJFWC*** is our heuristic meant to be used on computation-homogeneous platform.

SLJF and *SLJFWC* were initially built to work with off-line models, because they need to know the total number of tasks to perform at their best. So we transform them for on-line execution as follows: at the beginning, we start to compute the assignment of a certain number of tasks (the greater this number, the better the final assignment), and start to send the first tasks to their assigned processors. Once the last assignment is done, we continue to send the remaining tasks, each task being sent to the processor that would finish it the earliest. In other words, the last tasks are assigned using a list scheduling policy.

6.2 Experimental platform.

We build a small heterogeneous master-slave platform with five different laptops running the Linux operating system and connected to each other by a fast Ethernet switch (100 Mbit/s). The five machines are all different, both in terms of the amount of available memory and in terms of CPU speed (four have a CPU speed between 1.2 and 1.6 GHz, the last and oldest one has a processor Mobile Pentium MMX 233 MHz). The heterogeneity of the communication links is mainly due to the differences between the network cards (the oldest and slowest machine has a 10 Mbit/s network card, the four others have 100 Mbit/s ones). Each task will be a matrix, and each slave will have to calculate the determinant of the matrices that it will receive. Whenever needed, we play with matrix sizes so as to achieve more heterogeneity or on the contrary some homogeneity in the CPU speeds or communication bandwidths. We proceed as follows: in a first step, we send one single matrix to

each slave one after the other, and we calculate the time needed to send this matrix and to calculate its determinant on each slave. Thus, we obtain an estimation of c_i and w_i, according to the matrix size. Then we determine the number of times this matrix should be sent (n_{c_i}) and the number of times its determinant should be calculated (n_{w_i}) on each slave in order to modify the platform characteristics so as to reach the desired level of heterogeneity. Then, a task (matrix) assigned on P_i will actually be sent n_{c_i} times to P_i (so that $c_i \leftarrow n_{c_i}.c_i$), and its determinant will actually be calculated n_{w_i} times by P_i (so that $w_i \leftarrow n_{w_i}.w_i$).

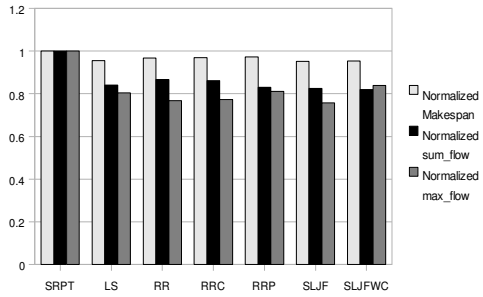
The experiments are as follows: for each diagram, we create ten random platforms, possibly with one prescribed property (such as homogeneous links or processors) and we execute the different algorithms on it. Our platforms are composed with five machines P_i with c_i between 0.01 s and 1 s, and w_i between 0.1 s and 8 s. Once the platform is created, we send one thousand tasks on it, and we calculate the makespan, sum-flow, and max-flow obtained by each algorithm. After having executed all algorithms on the ten platforms, we calculate the average makespan, sum-flow, and max-flow.

6.3 Results

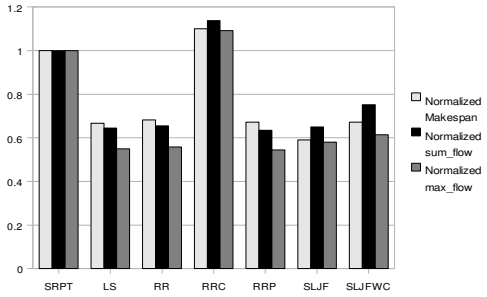
In the following figures, we compare the seven algorithms: *SRPT*, *List Scheduling*, the three *Round-Robin* variants, *SLJF*, and *SLJFWC*. For each algorithm we plot, from left to right, its normalized makespan (in blue), sum-flow (in green), and max-flow (in brown). We normalize everything to the performance of *SRPT*, whose makespan, max-flow, and sum-flow are therefore set equal to 1.

First of all, we consider fully homogeneous platforms. Figure 5(a) shows that all static algorithms perform equally well on such platforms, and exhibit better performance than the dynamic heuristic *SRPT* (heuristic 1). On communication-homogeneous platforms (Figure 5(b)), we see that *RRC* (heuristic 4), which does not take processor heterogeneity into account, performs significantly worse than the others; we also observe that *SLJF* (heuristic 6) is the best approach for makespan minimization. On computation-homogeneous platforms (Figure 5(c)), we see that *RRP* (heuristic 5) and *SLJF* (heuristic 6), which do not take communication heterogeneity into account, perform significantly worse than the others; we also observe that *SLJFWC* is the best approach for makespan minimization. Finally, on fully heterogeneous platforms (Figure 5(d)), the best algorithms are *LS* (heuristic 2) and *SLJFWC* (heuristic 8). Moreover, we see that algorithms taking communication delays into account actually perform better. We underline here that taking into account the communication heterogeneity is more important than the computation

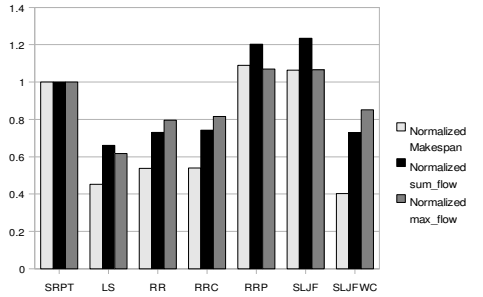
heterogeneity for a scheduler. This could be explained mainly because of the one-port model.



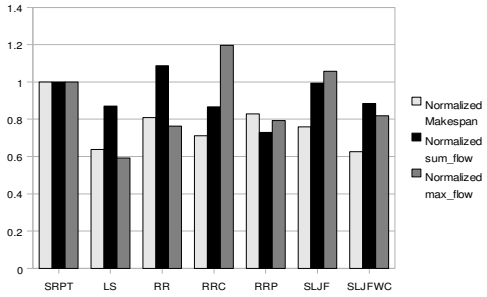
(a) Homogeneous platforms



(b) Platforms with homogeneous communication links



(c) Platforms with homogeneous processors



(d) Fully heterogeneous platforms

Figure 5. Comparison of the seven algorithms on different platforms.

In another experiment, we try to test the robustness of the algorithms. We randomly change the size of the matrix sent by the master at each round, by a factor of up to 10%. Figure 6 represents the average makespan (respectively sum-flow and max-flow) compared to the one obtained on the same platform, but with identical size tasks. Thus, we see that our algorithms are quite robust for makespan minimization problems, but not as much for sum-flow or max-flow problems.

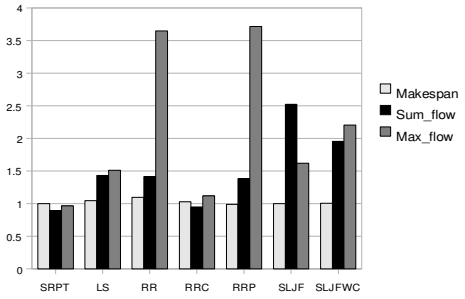


Figure 6. Assessing algorithm robustness: ratios of performance with varying task sizes compared to performance with identical task sizes.

7 Conclusion

In this paper, we have dealt with the problem of scheduling independent, same-size tasks on master-slave platforms. We enforce the one-port model, and we study the impact of heterogeneity on the performance of on-line and off-line scheduling algorithms as well as the impact of the communications on the design and analysis of the proposed algorithms. The major contribution of this paper lies on the theoretical side, and is well summarized by Table 1 for on-line scheduling. We have provided a comprehensive set of lower bounds for the competitive ratio of any deterministic on-line scheduling algorithm, for each source of heterogeneity and for each target objective function. An important direction for future work would be to see which of these bounds can be met, if any, and to design the corresponding approximation algorithms. We also have derived several new results for off-line scheduling with release dates, as an optimal makespan-minimization algorithm for communication-homogeneous platform, and a NP-hard proof of the scheduling problem on fully heterogeneous platform. Another important direction for future work would be to derive an optimal algorithm or to prove the NP-hardness for off-line scheduling with release dates on computation-homogeneous platforms.

On the practical side, we have to widen the scope of the MPI experiments. An extensive comparison of all the heuristics that we have implemented needs to be conducted on significantly larger platforms (with several tens of slaves). Such a comparison would, we believe, further demonstrate the superiority of those heuristics which fully take into account the relative capacity of the communication links.

References

[1] M. Adler, Y. Gong, and A. L. Rosenberg. Optimal sharing of bags of tasks in heterogeneous clusters. In *15th ACM Symp. on Parallelism in Algorithms and Architectures (SPAA'03)*, pages 1–10. ACM Press, 2003.

[2] M. Banikazemi, V. Moorthy, and D. K. Panda. Efficient collective communication on heterogeneous networks of workstations. In *Proceedings of the 27th International Conference on Parallel Processing (ICPP'98)*. IEEE Computer Society Press, 1998.

[3] M. Banikazemi, J. Sampathkumar, S. Prabhu, D.K. Panda, and P. Sadayappan. Communication modeling of heterogeneous networks of workstations for performance characterization of collective operations. In *HCW'99, the 8th Heterogeneous Computing Workshop*, pages 125–133. IEEE Computer Society Press, 1999.

[4] Amotz Bar-Noy, Sudipto Guha, Joseph (Seffi) Naor, and Baruch Schieber.

Message multicasting in heterogeneous networks. *SIAM Journal on Computing*, 30(2):347–358, 2000.

[5] Olivier Beaumont, Vincent Boudet, and Yves Robert. A realistic model and an efficient heuristic for scheduling with heterogeneous processors. In *HCW'2002, the 11th Heterogeneous Computing Workshop*. IEEE Computer Society Press, 2002.

[6] Olivier Beaumont, Arnaud Legrand, and Yves Robert. A polynomial-time algorithm for allocating independent tasks on heterogeneous fork-graphs. In *ISCIS XVII, Seventeenth International Symposium On Computer and Information Sciences*, pages 115–119. CRC Press, 2002.

[7] Olivier Beaumont, Arnaud Legrand, and Yves Robert. The master-slave paradigm with heterogeneous processors. *IEEE Trans. Parallel Distributed Systems*, 14(9):897–908, 2003.

[8] P.B. Bhat, C.S. Raghavendra, and V.K. Prasanna. Efficient collective communication in distributed heterogeneous systems. *Journal of Parallel and Distributed Computing*, 63:251–263, 2003.

[9] H. Casanova and F. Berman. *Grid Computing: Making The Global Infrastructure a Reality*, chapter Parameter Sweeps on the Grid with APST. John Wiley, 2003. Hey, A. and Berman, F. and Fox, G., editors.

[10] P. Chrétienne, E. G. Coffman Jr., J. K. Lenstra, and Z. Liu, editors. *Scheduling Theory and its Applications*. John Wiley and Sons, 1995.

[11] Pierre-François Dutot. Complexity of master-slave tasking on heterogeneous trees. *European Journal of Operational Research*, 164(3):690–695, August 2005. Special issue on Recent Advances in Scheduling in Computer and manufacturing Systems (J. Blazewicz, K. Ecker, and D. Trystram editors).

[12] H. El-Rewini, H. H. Ali, and T. G. Lewis. Task scheduling in multiprocessing systems. *Computer*, 28(12):27–37, 1995.

[13] M. R. Garey and D. S. Johnson. *Computers and Intractability, a Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

[14] L. Hollermann, T. S. Hsu, D. R. Lopez, and K. Vertanen. Scheduling problems in a practical allocation model. *J. Combinatorial Optimization*, 1(2):129–149, 1997.

[15] T. S. Hsu, J. C. Lee, D. R. Lopez, and W. A. Royce. Task allocation on a network of processors. *IEEE Trans. Computers*, 49(12):1339–1353, 2000.

[16] S. Khuller and Y.A. Kim. On broadcasting in heterogenous networks. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1011–1020. Society for Industrial and Applied Mathematics, 2004.

[17] J.K. Lenstra, R. Graham, E. Lawler, and A.H. Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.

[18] Joseph Y-T. Leung and Hairong Zhao. Minimizing mean flowtime and makespan on master-slave systems. *J. Parallel and Distributed Computing*, 65(7):843–856, 2005.

[19] P. Liu. Broadcast scheduling optimization for heterogeneous cluster systems. *Journal of Algorithms*, 42(1):135–152, 2002.

[20] J.M. Moore. An n job, one machine sequencing algorithm for minimizing the number of late jobs. *Management Science*, 15(1), September 1968.

[21] M. G. Norman and P. Thanisch. Models of machines and computation for mapping in multicomputers. *ACM Computing Surveys*, 25(3):103–117, 1993.

[22] J. M. Orduna, F. Silla, and J. Duato. A new task mapping technique for communication-aware scheduling strategies. In T. M. Pinkston, editor, *Workshop for Scheduling and Resource Management for Cluster Computing (ICPP'01)*, pages 349–354. IEEE Computer Society Press, 2001.

[23] Jean-François Pineau, Yves Robert, and Frédéric Vivien. The impact of heterogeneity on master-slave on-line scheduling. Research Report 2005-51, LIP, ENS Lyon, France, October 2005. Available at `graal.ens-lyon.fr/~yrobert/`.

[24] Jean-François Pineau, Yves Robert, and Frédéric Vivien. Off-line and on-line scheduling on heterogeneous master-slave platforms. Research Report 2005-31, LIP, ENS Lyon, France, July 2005.

[25] Jean-François Pineau, Yves Robert, and Frédéric Vivien. Off-line and on-line scheduling on heterogeneous master-slave platforms. In *Proceedings of the 14th Euromicro Conference on Parallel, Distributed and Network based Processing (PDP 2006)*, Montbéliard-Sochaux, France, February 15-17 2006.

[26] Kirk Pruhs, Jiri Sgall, and Eric Torng. On-line scheduling. In J. Leung, editor, *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, pages 15.1–15.43. CRC Press, 2004.

[27] C. Roig, A. Ripoll, M. A. Senar, F. Guirado, and E. Luque. Improving static scheduling using inter-task concurrency measures. In T. M. Pinkston, editor, *Workshop for Scheduling and Resource Management for Cluster Computing (ICPP'01)*, pages 375–381. IEEE Computer Society Press, 2001.

[28] T. Saif and M. Parashar. Understanding the behavior and performance of non-blocking communications in MPI. In *Proceedings of Euro-Par 2004: Parallel Processing*, LNCS 3149, pages 173–182. Springer, 2004.

[29] J. Sgall. On line scheduling-a survey. In *On-Line Algorithms*, Lecture Notes in Computer Science 1442, pages 196–231. Springer-Verlag, Berlin, 1998.

[30] B. A. Shirazi, A. R. Hurson, and K. M. Kavi. *Scheduling and load balancing in parallel and distributed systems.* IEEE Computer Science Press, 1995.

[31] Barbara Simons. Multiprocessor scheduling of unit-time jobs with arbitrary release times and deadlines. *SIAM Journal on Computing*, 12(2):294–299, 1983.

[32] Oliver Sinnen and Leonel Sousa. Communication contention in task scheduling. *IEEE Trans. Parallel Distributed Systems*, 16(6):503–515,

2004.

[33] M. Tan, H. J. Siegel, J. K. Antonio, and Y. A. Li. Minimizing the aplication execution time through scheduling of subtasks and communication traffic in a heterogeneous computing system. *IEEE Transactions on Parallel and Distributed Systems*, 8(8):857–871, 1997.